iCAP: Rapid Prototyping of Context-Aware Applications

Timothy Sohn, and Anind Dey

IRB-TR-03-016

October 2003

Proceedings of the CHI 2004 ACM Conference on Human Factors in Computing Systems

Intel Research Berkeley

iCAP: Rapid Prototyping of Context-Aware Applications

Timothy Sohn
Dep't of Computer Science and Engineering
University of California, San Diego
La Jolla, CA USA
tsohn@cs.ucsd.edu

Anind K. Dey
Intel Research, Berkeley
Berkeley, CA USA
anind@intel-research.net

ABSTRACT
Although numerous context-aware applications, those that implicitly take their context of use into account, have been developed, and advances have been made in technology to acquire contextual information, it is still difficult to develop and prototype interesting context-aware applications. This is largely due to the lack of programming support available to both programmers and end users. In this paper we focus on providing support to end users to open up the space of context-aware application design to a larger group of users. We present iCAP, the interactive Context-Aware Prototyper, a system that allows end users to visually design a wide variety of context-aware applications, including those based on if-then rules, temporal and spatial relationships and environment personalization. iCAP allows users to quickly prototype and test their applications without writing any code. We describe the system design and several applications that demonstrate iCAP's richness and ease of use. We also describe results from a user study performed with 15 end users and 5 programmers that illustrates iCAP's expressiveness and usability.

Author Keywords
Context-aware, end user programming, visual prototyping

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces – prototyping; D.1.7 [Programming Techniques]: Visual Programming; General Terms: Human Factors, Design; Keywords: end-user programming

INTRODUCTION
In the past several years, there has been an increased effort and interest in building and deploying context-aware applications. Users' environments contain context information that can be sensed by an application, including location, identity, activity, and the state of nearby people. Context-aware computing involves sensing this context to implicitly provide information and services appropriate to a user's context. Many groups have developed infrastructures and toolkits to support the next era of ubiquitous computing [26], however few have focused on empowering end users in building context-aware applications [9,13]. Currently, developing a context-aware application requires developers and end users alike to either build their application from scratch (involving painful direct interaction with hardware sensors and devices), or to use an enabling toolkit. Even with low-level toolkit support for acquiring context [4,7,11,24], individuals still must write large amounts of code to develop simple sensor-rich applications.

This inhibits the design of interesting applications, especially for end users, who end up having little control over how these applications behave. Instead, end users with little technical expertise should be able to exercise control over context-aware systems and rapidly prototype applications. They have more intimate knowledge about their activities and environments than a hired programmer and they need the ability to create and modify applications as those activities and environments change. Without such ability, these applications acting implicitly could significantly annoy users as they fail to meet users' needs.

To address these issues we built the interactive Context-aware Application Prototyper (iCAP), a system aimed at lowering barriers for end users to build interesting context-aware applications for their instrumented environments, *by not requiring them to write any code*. In particular, iCAP allows a user to describe a *situation* and associate an *action* with it. iCAP is a visual, rule-based system that supports prototyping of 3 common types of context-aware behaviors: simple if-then rules, relationship-based actions and environment personalization. In supporting prototyping without writing code, iCAP allows end users to exert control over sensing systems and dictate the behavior of their applications.

The most common context-aware applications are described naturally as a collection of rule-based conditions [10]. An example is a museum tour guide [1], where the rule is "when a user stands in front of a particular exhibit, show them content related to that exhibit." Another canonical example is a smart home [17], with a simple rule being "turn on the lights whenever someone is in the room." The iCAP system is designed upon this rule-based paradigm with two separate parts: a visual rule building interface and an underlying rules engine. The visual interface, shown in Fig. 1, allows users to build, prototype, deploy and test their

application. It also separates users from the difficulties dealing with low-level sensor input or toolkit-level details. The rules engine is a database of user-defined rules that receives and sends events to evaluate and execute each rule in support of an application. Context inputs to trigger the rules can either come from a Wizard of Oz interface manipulated by the user [6], or from a real instrumented context-aware environment.

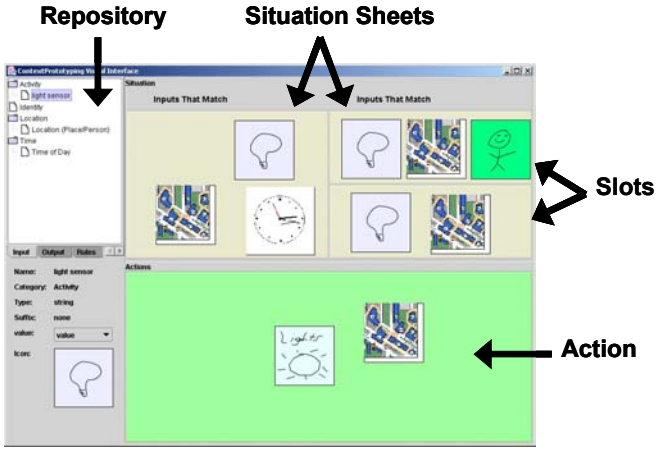

Fig. 1. The iCAP user interface has two main areas: a tabbed window on the left that acts as a repository for user-defined inputs, outputs, locations, people and rules and the situation area on the right where these components can be dragged to construct a rule. The rule shown uses 2 situation sheets, where one (on the right) is split into 2 slots. The rule being prototyped is: *when* the lights are on in the kitchen past 10pm, or a user is in the bedroom with the lights on and the lights are on in the kitchen, *then* turn off the lights in the kitchen.

iCAP supports three common types of context-aware applications. The first type, *simple if-then rules*, consist of IF-THEN rules where an action is triggered when a condition is satisfied. As discussed earlier (with the tour guide example), many basic applications can be described in this way. The next type is *relationship-based actions.* Humans are naturally relational, and think in terms of *personal, spatial* and *temporal* relationships. For instance, I am aware that my roommate entered the living room five minutes ago, and that my room and his bedroom are connected by a hallway. When the system recognizes that my roommate is in the next room, it can prompt me to ask him about going grocery shopping. iCAP provides the necessary support to develop rules that are built on these types of relationships. Lastly, it supports *environment personalization*, where an environment satisfies the differing preferences of its occupants. For instance, one user may enjoy bright lights and rock music, while another prefers dim lights with classical music. To satisfy both users, the environment must account for their preferences and adjust itself accordingly.

The next section surveys previous research in the areas of context-aware computing, rule-based systems, and informal interfaces, providing further motivation for our work. In the following sections we present an example scenario and describe the ease of interaction in building context-aware applications through the visual interface. We then describe the necessary underlying support required in the rules engine. We demonstrate the benefits of iCAP through a number of demonstration applications built with the system that cover an important design space in context-aware computing. We show that iCAP is a usable and useful system through a user study we conducted. For usability and complexity reasons, most systems designed for end users are more functionally constrained than systems designed for programmers and iCAP is no exception. We therefore conclude with a discussion of the limitations of iCAP and future directions for this research.

RELATED WORK

This work has been informed by research in the areas of context-aware computing and visual rule-based systems for end users. We discuss both of these here.

Context-Aware Computing

Since Weiser's vision of ubiquitous computing more than a decade ago [26], many groups have explored the domain of context-aware applications. Schilit's ParcTab system first began to shape the infrastructure and applications that could be built to support context-awareness [23]. Soon after this, many architectures and applications such as stick-e notes [19] and GeoNotes [8] focused on allowing end users to contextually share data with each other by placing virtual objects in a context-aware environment. However, many of these types of applications were being written from scratch, which has a high development cost [20]. One must interface to an external sensing system, gather the appropriate sensor data, develop a rule-based engine, and execute the desired actions. With these architectures, there is little support for rapidly prototyping applications.

The Context Toolkit is one step in the direction of eliminating the high development costs associated with context-aware applications and providing reusable code to allow easy prototyping [7]. The Context Toolkit provides the sensing system, and returns desired data to the application developer for use in providing context-enhanced services and information. Other similar context sensing infrastructures include the Context Fabric [11], Cooltown [4], SOLAR [3], and TEA [24]. However, even with the low-level sensing support that each of these provides, there is still a vast amount of work required by programmers to build applications based on simple rules. Programmers still must maintain a database of rules, trigger different actions, and provide some means of prototyping the application. In essence, existing systems do not provide any real support for prototyping by programmers and definitely not end-users, thus demonstrating a need for a system like iCAP.

Visual Rule-Based Systems

We chose to make iCAP a visual environment for users to prototype context-aware applications for reasons of simplicity and intuitiveness. Visual programming languages

have proven effective in taking advantage of user's spatial reasoning skills [25]. They involve interactively laying out building blocks and defining relationships among them. This programming style is not only simple and effective for many types of users, but is also especially intuitive for end users. When applied to a new domain like context-aware computing, visual programming provides tools needed to allow creative end-users to easily build novel applications.

Mackay, Malone *et al.* showed in the Information Lens project that people with little computer experience could create and use rules effectively [15]. Using a rule-based system [10], we have built a visual environment to make the design of context-aware rules as simple as possible. Agentsheets capitalizes on the idea of visual rule-based programming by allowing end users to establish relationships among different autonomous agents [21]. Although it could be expanded to support context-aware applications, Agentsheets still requires a high level of expertise to use [22], and supports limited sensing and actuation. In contrast, we aim to provide even novices with support to build context-aware applications. The Jigsaw Editor supports end-user reconfiguration of home devices using a novel jigsaw puzzle metaphor [13]. The creators state that they "do not seek the richness of programming expression allowed by iCAP" [13]. Our goal is to support the building of expressive applications by novice users, trading off some learnability for this expressiveness. The Alfred system uses recordable speech-based macros to support users in building applications for a smart environment [9]. However, Alfred focuses on rules based on explicit user interaction (pressing a button, speaking a phrase), and, unlike iCAP, does not support conditions based on contextual cues. Successful rule-based visual programming languages in other domains have provided inspiration, including Stagecast Creator™ and KidSim for gaming [5].

Summary

Clearly, there is a need for a context-aware prototyping environment that enables end users to build rule-based context-aware applications. By building upon work in visually-based rule systems, we can address this to provide an effective prototyping tool, empowering end users to build interesting context-aware applications that covers an important design space in context-aware computing (discussed in our validation section).

The next section presents an example of how one uses iCAP's visual interface to build a context-aware application and demonstrates the usefulness of the tool. We will revisit this example as we describe the details of the visual interface and the underlying rules engine.

EXAMPLE APPLICATION

One can imagine a buddy alert application where a user, John, wants to be notified when his friends are in a location adjacent to his. His rule would be:

IF I am in location1 AND my friend is in an adjacent location location2 THEN beep my cell phone.

The user first creates the people and artifacts involved in the rule and adds them to the repository. For each person, the user sketches an icon that will be associated with that person, and specifies that it is part of a "friends" group (Fig. 2a). He then creates a new output, the cell phone and specifies the name of the output (phone), the type of output (binary: ON/OFF, for simplicity), and a sketch/image of the output (Fig. 2b). It is now ready to use in designing rules by simply dragging the appropriate icon onto the rule sheets.

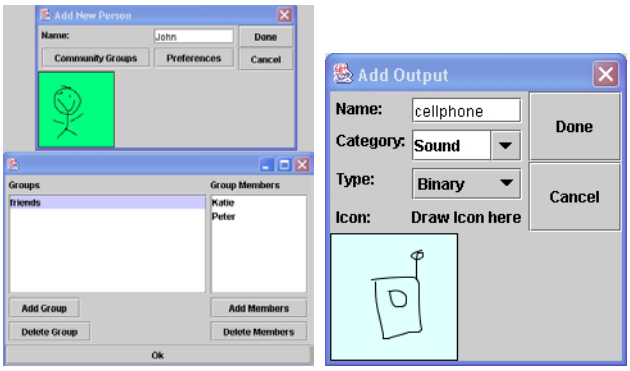

Fig. 2. (a) Creation of a person and his personal groups. (b) Creation of the cellphone.

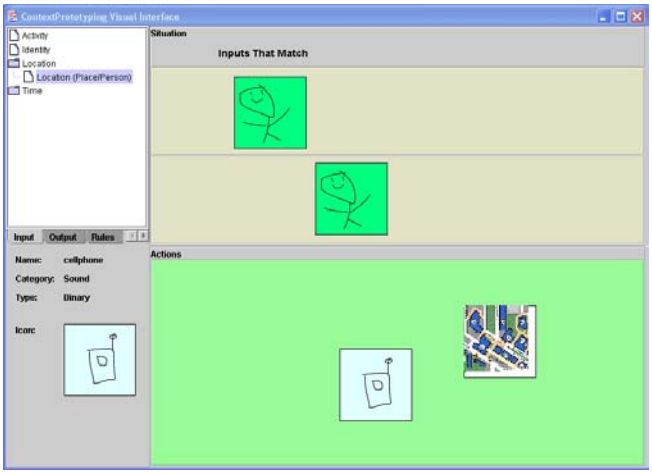

Fig. 3. Rule layout for the example application: if my friend and I are in adjacent locations, then beep my cell phone.

The user selects "New Rule" using a pie menu interactor and the system creates one visual area, or *sheet*, on top in beige for entering the situation (IF) and another sheet on the bottom in green for the action (THEN) (Fig. 3). The example situation has two conditions that are specified by laying out icons: "John is in location1" and "any friend of John's is in location2". The user splits the single input sheet into two vertical "slots" that will be related by a conjunction. In the first slot, he drags in the icon of John from our set of inputs. In the second slot, he again drags in the object of John, but specifies that it represents John's friends. Since he does not specify locations for either slot, the system automatically assigns them variable locations.

To specify the action, the user drags in the cell phone icon from our set of outputs to the action sheet, he sets the action to be "turn on". The location of this device should be John's location, so he drags in a Location object (map icon), and sets that to be John. Thus the action sheet has two icons that, together, specify the action "turn on John's phone".

Finally, the rule is saved and the user is shown a drop down menu to resolve any relations between the variable location values. He specifies an adjacency relationship, indicating that "John's location should be adjacent to his friend's location". The rule is saved and appears in the rule panel on the left. He can test his buddy alert application by selecting prototype from the left pane. This launches another window where he can either control (simulate) the relevant context inputs or connect to an existing context infrastructure, and see if his rule behaves as expected (Fig. 4).

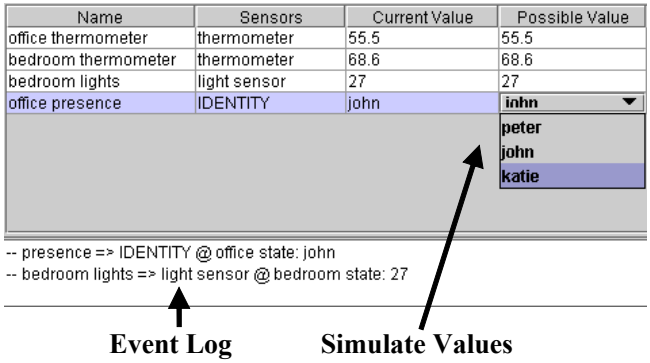

Fig. 4. Prototyping mode where users can simulate values and see event changes throughout the system in the event log.

THE ICAP SYSTEM

The previous section illustrated how a user could use the iCAP system to build a context-aware rule. Here, we describe the design of iCAP itself. It is a rule-based system consisting of two main pieces (both implemented in Java): a visual interface for building rules and a rules engine that stores the built rules and evaluates them when the rule is being run. The iCAP visual interface (shown above) provides a simple way for users to design and prototype context-aware applications. The user only needs to deal with defining inputs and outputs and using these to visually create situations and associated actions, or if-then rules. There is no explicit coding involved, which is a tremendous benefit for non-programmers because it allows them to easily explore the context-aware computing design space.

We iterated on the design of the user interface multiple times, starting with paper prototypes through to the final interface which we present here. At each stage, we obtained feedback from local experts and test subjects. The interface has one window with two main areas (Fig. 1), a tabbed window on the left that is the repository for the user-defined inputs, outputs, locations, people, and rules, and, on the right, a rules area where these components can be dragged to construct a conditional rule. The rules area is split into two areas, one (top) for situations (IF) and one (bottom) for actions (THEN). We built iCAP on top of SATIN, a toolkit for building sketching-based applications [12]. Through SATIN, we provide pie menus to better support pen interaction [2], and gesture support for issuing common commands such as cut, copy and paste of graphical objects.

iCAP Interaction

Interaction with iCAP has three steps. First, the user sketches inputs and outputs relevant to their rules (if they do not already exist). Users can also define person objects with personal groups and preferences as shown in the example scenario, as well as location objects to facilitate adjacency comparisons. Second, these elements are dragged and composed to create rules. Finally, the entire set of rules can either be simulated or connected to a live context infrastructure using the prototyping mode.

Creating Inputs and Outputs

Inputs are used to create situations (IF) and outputs are used to create actions (THEN). Each input and output component is associated with a user-sketched graphical icon. The icons are colored differently depending on whether it is an input or an output. Each input contains a set of potential values, a unit (*e.g.* degrees Celsius for temperature), and type (*e.g.* integer, string), An input's potential values can be provided as a range ([1-100]) or list (1;7;10). For example, a temperature sensor would have the name field 'temperature', the units 'degrees Celsius', the type 'integer', and the potential values would be [-10-40] indicating a range between -10 and 40.

Outputs are created in the same manner as inputs, but contain different parameters to specify. Each output is either a binary (on/off) or gradient (range from 1-10) device. In addition, there are 3 pre-defined categories an output device is associated with: lighting, sound and temperature. These categories (along with user-defined categories) are used in personalizing an environment and changing the appropriate device for a user's preference (*e.g.* sound device for sound preferences). An example output was the cell phone in the buddy alert application discussed above. In addition, outputs have a content field used to simulate output modalities. For example, with a music player, we could fill in the content field with "Beethoven Symphony", which would output that string when turned on, to simulate the playing of a classical music piece.

Creating People and Locations

Person and Location objects are created similarly to inputs and outputs, as described in the example scenario. These objects are essential for relationship and personalization rules. Person objects are created with a name and optional fields for preferences as well as community groups. The system recognizes three predefined preferences for a person: lighting, sound and temperature. It controls devices in the user's room to match the preferences of its occupants. Users can also define custom preferences such as a music category with a preference being classical music. Person

objects can also be created with community groups such as friends or family. Each Person object can be a member of multiple groups. These groups allow the creation of general rule structures such as "if a family member…".

Location objects are created to specify that a condition or action must take place in a particular location. They simply require a name upon creation. Optionally, a user can indicate what locations are connected together, allowing the creation of rules that take advantage of these spatial adjacencies. In addition, the user can specify whether environment personalization should be turned on for this location. If turned on, the location will attempt to satisfy the preferences of its occupants as described above.

Constructing Rules: Simple if-then rules, Relationship-based Actions and Environment Personalization

Users create these objects and use them to define rules. iCAP supports the construction of simple if-then rules, spatial, temporal and personal relationship-based actions and environment personalization. Users build these rules by dragging and dropping inputs and outputs onto the situation and action sheets for each rule. After dragging each icon, the user needs to specify conditions governing the behavior of the input, including <, <=, >, >=, = and combinations to form ranges.

To support users in visually specifying Boolean logic in *simple if-then* rules, we implemented Pane and Myers' matching scheme [18]. Their matching layout uses the vertical dimension with a label "inputs that match" to represent the AND operator and the horizontal dimension for the OR operator. They showed that this layout provides a simple, intuitive way to express Boolean logic, for non-programmers, including both children and adults. To support this, our system uses two important metaphors, "sheets" and "slots". All components on a single sheet are related to each other by conjunction. A sheet can be split into multiple vertical slots that are related to each other by a conjunction (AND). Users can add multiple horizontal sheets which are related by a disjunction (OR). Fig. 1 illustrates the use of sheets and slots, describing a rule that turns off the lights in the kitchen if it is past 10pm (left sheet), OR if someone is already in the bedroom and (right sheet, top slot) with the lights turned on AND the lights are on in the kitchen (right sheet, bottom slot).

People naturally specify conditions in general terms such as "the temperature in this room is greater than the temperature in another room", or use general terms like "anyone", or "anywhere". These phrases motivate the need to support a generalization for specifying rules and not just simple fixed values. We call these general values *variable values*. If we look at the initial scenario presented, variable values were used to design a *spatial relationship* rule that has two people in different locations. John is in variable_location1 and Peter is in variable_location2. There is a comparison included in the rule that variable_location1 *adjacent to* variable_location2. In addition to variable locations, variable values can be assigned to person objects to create expressive rules involving "anyone".

Relative *temporal relationships* such as "before" and "after" are also supported for creating rules. Recall the slots used in creating extra AND relationships on a single input sheet. These slots can be overloaded to represent an ordering of time, so the first slot represents the first event, the second slot represents an event happening after the first one, and so on. The situation would then be satisfied if these events happened in the desired order. Objects can be set to keep track of a certain time period of activity (*e.g.* 5 minutes before) or a relative time period (*e.g.* the next person who walks in). These temporal relationships further exhibit the power of iCAP in building conditional rules.

Personal relationships and *environment personalization* are tightly integrated together. Personal relationships are supported through the use of community groups, as illustrated in our initial scenario about a group of friends. Examples include family, friends and co-workers. We support personalization by allowing individuals to develop preferences and community groups. The system, by default, supports preferences for lighting, sound, and temperature. In addition, the user can declare other preferences such as type of music that she wants to be played wherever she is in her smart environment. By setting a flag for a location, a user can indicate whether the environment should take these preferences into account and change itself when users are in that location. When a person enters a room (with a set flag), the location analyzes the preferences of each person present and tries to satisfy these preferences. Combinations of personal relationships and personalization are supported, for example, in combining the preferences of all my family members. Aggregation of preferences to a single result is performed by following a set of heuristics. At this point, new heuristics can only be added by writing code, but we support a number of common heuristics (*e.g.* oldest person wins and person who has been in the room the longest wins) and plan to allow end users to visually create their own.

Running the Application

After a number of rules have been defined, the entire rule set can be tested using the iCAP rules engine. The engine can either be set to simulate the context-aware environment, or be used in conjunction with a real context-aware environment, the Context Toolkit. Users can interact with the engine in simulation to change the value of defined inputs and evaluate the behavior of the rules being tested (Fig. 4). In simulation mode, outputs are visually updated onscreen, whereas with a real environment, real outputs in the physical environment are updated. With the engine, users can quickly design and test their applications, without having to create an entire infrastructure for collecting or simulating context and without writing any code.

Rules engine

The rules engine sits between the visual interface and a real or simulated context-aware infrastructure. It is a repository of rules accepting inputs and triggering outputs. Rules are represented in the engine as a situation and associated action, with each represented as a Boolean tree. Non-leaf nodes represent Boolean operators (*e.g.* AND, OR) and leaf nodes contain comparisons (*e.g.* John is in the bedroom or temperature > 15°) or actions (*e.g.* call Anne's pager).

The rules engine supports all general comparison operations (*e.g.* =, >, <, *etc.*), as well as relative temporal, spatial and personal relationships for evaluating rules, as described in the preceding sections. Evaluation of rules is based on context input received from either a Wizard-of-Oz interface (Fig. 4), and/or a real context environment. For the latter, the engine provides an interface to the Context Toolkit, to support the passing of events from the real environment to the rules engine and to support the rules engine in executing actions in the real environment. Leaf nodes in our Boolean trees act as queries to a discovery service in the Context Toolkit, enabling them to bind to real-world sensors and actuators in a user's smart environment. The engine can either be set to operate only in a simulated environment, where the user has complete control over the inputs passed into the system, or to map as many inputs and outputs as possible to real context provided by the Context Toolkit and simulate the remaining unmapped ones. In this latter situation, the user only has simulation control over inputs unavailable in the real context environment.

VALIDATION

In the preceding sections, we have motivated and described the design of iCAP. Here, we validate iCAP in two ways. First, we describe the results of a user study we performed on iCAP demonstrating its usability. Second, we show how iCAP covers a design space of context-aware applications outlined by Schilit and how iCAP enables end users to build canonical applications taken from the literature.

User Study

To better understand the usability of iCAP, we had 15 end users, experienced with computers but with little to no programming experience, and 5 experienced programmers use the system and give us feedback. All users were college graduates or current students in their 20's. Most users were slightly familiar with the topic of smart homes (euphemism for context-aware), but none had ever built a smart home application. We gave them a 10-minute tutorial of iCAP, covering four types of rules (simple IF-THEN, personal, spatial and temporal relationships) that it supports and asked them to build 8 fixed rules (2 each of the four rule types) and 2 open design rules of their choice.

Analysis

The user study shows that iCAP is a usable system. Users were successful in using iCAP to create simple and complex rules. All the users were able to create all of the 8 fixed rules in a reasonable amount of time. Table 1 shows the average times for both end users and programmers in completing each of the 8 standard tasks.

Beyond this success, there were four noticeable trends in the user studies. First, there was clearly a learning curve in using iCAP. Overall, users took less time in completing the second rule of a given type than the first rule.

Second, programmers were faster than end users in creating rules. In general, both groups had similar difficulties creating rules, but, from our observations, programmers tended to recover faster from problems in using iCAP.

Type	Rule	End User	Programmer
S	If John is in the bedroom, turn on the lights in the bedroom.	3.93 (1.22)	2.8 (1.3)
S	If someone is at the country music exhibit, play Garth Brooks music.	3.53 (1.88)	2.6 (1.34)
P	If any of Joe's friends are in the office, page Joe.	3.73 (1.49)	3.6 (0.89)
P	If Joe's friends are in the office, turn down the music in the office.	2.4 (0.74)	2.2 (1.1)
L	If Peter is in an adjacent room to Joe, page Peter.	8.13 (2.17)	4.2 (1.48)
L	If Peter is in a room and the lights are on, and the lights are on in an adjacent room, turn the lights off in that adjacent room.	6.33 (1.8)	4.2 (0.84)
T	If your 5-year old child, Kevin enters the bathroom and then enters the bedroom and the faucet has not been used, sound a signal to tell Kevin to wash his hands.	5.87 (2.85)	5.6 (2.7)
T	If Ed walks into the living room and Daniel walks in next, set the lights to be "full lights".	2.73 (1.22)	2.0 (0)

Table 1. Average rule completion time (and standard deviation), in minutes. S=Simple if-then; P=Personal; L=Location or Spatial; T=Temporal.

Third, spatial rules took the longest to create for both user groups, although programmers had an easier time than end users. Spatial rules inherently use variable values and this was a particularly difficult concept for the end users to grasp. Most users did not understand how to create a variable location. Instead of not providing a location icon to indicate a variable location, they created a single location called "room" or multiple locations called "room1" and "room2" and placed them on the situation sheet. Users that correctly created variable locations had trouble using drop-down menus to set the constraints necessary to resolve the variable location comparisons (*e.g.* location *1 *is adjacent to* location *2). To address both issues, we will revise iCAP using suggestions from our users: create "variable location" and "variable person" icons for users to use in rule creation, and provide gestures to specify constraints (*e.g.* grouping icons together, joining with lines).

Finally, all users found the Pane and Myers matching scheme [18] to be useful in visualizing 'AND' and 'OR'

relationships. Users were able to easily recall the vertical and horizontal mappings, respectively, for these two Boolean operators. We observed many instances where users were unsure whether to add a sheet or a slot, and by trial and error they were able to determine which one they wanted based upon the visual representation (*i.e.* horizontal sheets is OR, vertical slots is AND). In addition, while spontaneously thinking aloud, several users referred to the label on the sheet and said "it says inputs that match, so that means everything on this sheet is an AND relationship".

However, some users had trouble with the overloaded slot operator. In most cases, slots support ANDing two expressions together, but when used in temporal rules, they represent an ordering of events. This overloading confused 2 users, who added sheets instead of slots. One way to prevent this confusion is to label each sheet numerically implying order when developing a temporal rule.

Open Design Rules

In addition to the 8 fixed rules, each user was asked to build 2 rules of their choosing in iCAP. From our observations of users, iCAP was successful in opening up the design space for creative individuals to develop and specify their own rules. Many of the rules touched on scenarios, constraints and conditions that we had not envisioned nor discussed in the brief tutorial we gave the users, yet users were mostly successful in creating their rules.

Out of the 40 rules attempted (2 for each person), users successfully completed 25 rules covering all the types of rules that iCAP supports. Examples of rules created are:

Simple: *If lights turn on in the office, turn on the computer.*
Personal/Personalization: *If Kevin's friends are in the living room and Kevin enters the living room, then raise the volume of music and dim the lights in the living room*
Spatial: *If Tim's children are adjacent to the pool, page Tim*
Temporal: *If someone goes into the bathroom, walks out and the toilet hasn't been flushed, flush the toilet.*

Of the remaining 15 rules, 6 were rules that iCAP supports, but that users could not figure out how to create. An example is "If the pastor is looking at someone during sermon and someone is sleeping, shock someone". This rule can be created with iCAP but is quite complex.

The final 9 rules were ones that iCAP cannot currently support, illustrating the need for future extensions to iCAP. One user wanted to make a rule that involved specifying "John and Mary's son". This rule requires iCAP to support intersections between people and their community groups. Another user wanted to specify the rule "If an object exists in the bedroom to play music, turn it on". By providing an "existence" or discovery operator, we can support rules like this. A third user wanted to specify the condition "if no one is in the room", but iCAP does not support negation of conditions. We are exploring ways to address these issues.

Summary

Our evaluation of iCAP showed that it was successful in supporting users in building simple and complex rules. These rules spanned the range of simple if-then rules, environment personalization and personal, spatial and temporal relationship-based actions. Most of the users were successful in constructing rules in the open design portion of the study, creating rules quite different than those proposed by us. As stated earlier, iCAP trades off some learnability for complexity in the types of rules it can support, allowing users to create more expressive rules than either the Alfred system or the Jigsaw Editor [9,13]. While users had some issues with iCAP, all indicated that they would use it if hardware to outfit a smart home were readily available. One user said iCAP was "exactly what I imagined a visual programming interface would be like to develop these applications." Another user described iCAP as "easy to use" and "fun and exciting to be able to easily build applications for my home."

Context-Aware Design Space

As further validation for iCAP, we show that iCAP facilitates the building of a wide variety of context-aware applications, covering an important design space in context-aware computing and canonical applications taken from the literature. Schilit has 4 categories of applications: [23]:

Context-triggered actions, where IF-THEN rules are used to specify how a context-aware system should adapt.
Automatic contextual reconfiguration, where components are added, removed or the connections between them are altered.
Contextual information and commands, where different results are produced according to the current context.
Proximate selection, where objects located nearby are emphasized or made easier to choose.

Allowing users to visually create applications with context-triggered actions (if-then rules specifying how a context-aware system should adapt) has been the basis of our work in iCAP and we have described several examples. An example of automatic contextual reconfiguration taken from [23] is reconfiguration to share an object among people. iCAP supports this through environmental personalization where, for example, a room's lighting devices are adjusted to meet the preferences of its occupants. An example of contextual information and commands is Schilit's location browser that presents information relevant to a user's location. iCAP supports the delivery of user-selected content to fixed or mobile outputs (as described above). We leveraged this to build an application that delivers a list of people in adjacent rooms to a user's cell phone and updates this when the user enters a new location. Finally, we built an example of a proximate selection application that shows a list of the available outputs in the user's current location on fixed displays in the room.

Finally, we have used iCAP to build a variety of canonical context-aware applications taken from the literature. Common applications include tour and information guides

[1], reminder systems [16], and environmental controllers [17]. Users can build tour guide applications by creating a collection of rules that present user-defined content when someone enters a particular location, similar to Schilit's contextual information and command category. The content delivery can be customized based upon the profile or preferences of the user, to display information about the architecture of a space to a user who is interested in architecture, for example. A reminder system can be built in much the same way as a tour guide application, by creating a number of rules that deliver content, a reminder, when the user is in a particular situation. For example, when I am in a room adjacent to Katie, remind me to give her the book I borrowed from her. Finally, home automation systems or environment controllers are another popular context-aware application. As we described earlier, iCAP supports the creation of rules that control heating and lighting conditions and environment personalization when occupants of a space have differing preferences.

CONCLUSIONS AND FUTURE WORK

In this paper, we presented iCAP, the interactive Context-aware Application Prototyper. iCAP is a visual rule-based environment that supports end users in prototyping context-aware applications without writing any code. iCAP provides two main benefits: opening up the space of context-aware application design to a larger group of users than just programmers; and, it gives control of what should happen in a context-aware environment to the people it most affects, the end users.

iCAP supports users in designing and implementing a context-aware application, testing it under simulated and real conditions and revising it, as needed. In particular, it supports the creation of if-then (or situation-action) rules that are triggered by contextual cues, the building of spatial, temporal and personal relationship-based rules, and the building of environment personalization systems.

We validated its usefulness in two ways. First we obtained feedback from 15 end-users and 5 programmers who told us iCAP was a powerful system that they would like to use in the future. We then showed that iCAP could be used to build both canonical context-aware applications and applications that covered Schilit's design space.

Most systems designed for end users have more constrained functionality than systems designed for programmers. While we have used iCAP to build a wide variety of context-aware applications, it is not as expressive as existing programming systems for building applications [4, 7,11,24]. This includes supporting more sophisticated Boolean logic and the ability to activate and deactivate rules based on contextual cues. In addition, we would like to extend iCAP to support context-based retrieval systems that tag captured information with contextual cues to aid future retrieval [14]. iCAP can already capture context cues, so we would need to add the ability to store those cues persistently and attach them to user-provided content and provide a mechanism for querying the cues and content. We would like to increase the expressiveness of iCAP while, at the same time, maintaining its ease of use and increasing its learnability to improve users' performance at creating rules. We plan to redesign the interface and rules engine to address the issues that arose during our user study.

REFERENCES

1. Abowd, G.D. *et al.* Cyberguide: A mobile context-aware tour guide. *Wireless Networks* 3(5) 1997, 421-433.
2. Callahan, J. *et al.* An empirical comparison of pie vs. linear menus. *Proc. CHI 1988*, 95-100.
3. Chen, G. and Kotz, D. Solar: An open platform for context-aware mobile applications. *Proc*. *International Conference on Pervasive Computing 2002*, 41-47.
4. Cooltown homepage. http://cooltown.hp.com
5. Cypher, A. and Smith, D.C. KidSim: End user programming of simulations. *Proc. CHI 1995*, 27-34.
6. Dahlback,N. *et al*. Wizard of Oz studies – Why and how. *Proc. Intelligent User Interfaces 1993*, 193-200.
7. Dey, A.K. *et al*. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *HCI Journal*, 16(2-4) 2001, 97-166.
8. Espinoza, F. *et al.* Geonotes: Social and navigational aspects of location-based information systems. *Proc. UBICOMP 2001*, 2-17.
9. Gajos, K. *et al*. End user empowerment in human centered pervasive computing. *Proc. Pervasive 2002*, 134-140.
10. Hayes-Roth, F. Rule-based systems. In *Communications of the ACM* 1985, 921-932.
11. Hong, J.I. and Landay, J.A. An infrastructure approach to context-aware computing. *Human-Computer Interaction Journal*, 16(2-4) 2001, 287-303.
12. Hong, J.I. and Landay, J.A. SATIN: A toolkit for informal ink-based applications. *Proc. CHI 2000*, 63-72.
13. Humble, J., *et al*. 'Playing with your bits': user-composition of ubiquitous domestic environments. *Proc. UBICOMP 2003*, to appear.
14. Lamming, M. and Flynn, M. Forget-me-not: Intimate computing in support of human memory. *Proc. International Symposium on Next Generation Human Interfaces 1994*, 1125-128.
15. Mackay, W.E. *et al.* How do experienced Information Lens users use rules? *Proc. CHI 1989*. 211-216.
16. Marmasse, N. and Schmandt, C. Location-aware information delivery with commotion. *Proc. HUC 2000*, 151-171.
17. Mozer, M.C. The neural network house: An environment that adapts to its inhabitants. *Proc. AAAI Spring Symposium on Intelligent Environments 1988*, 110-114.
18. Pane, J.F. and Myers, B.A. Tabular and textual methods for selecting objects from a group. *Proc. IEEE Int'l Symposium on Visual Languages 2000*. 157-164.
19. Pascoe, J. The Stick-e Note Architecture: Extending the interface beyond the user. *Proc. Intelligent User Interfaces 1997*, 261-264.
20. Pascoe, J., *et al.* Issues in developing context-aware computing. *Proc*. *HUC 1999*, 208-221.
21. Repenning, A. and Citrin, W. Agentsheets: Applying grid-based spatial reasoning to human-computer interaction. *Proc*. *IEEE Symposium on Visual Languages 1983,* 77-82.

22. Scerri, P. and Reed, N. The EASE actor development environment. *Proc. Swedish AI Society 2000*.

23. Schilit, B. *et al.* Context-aware computing applications. *Proc. WMCSA 1994*.

24. Schmidt, A. and Van Laerhoven, K. How to build smart appliances. *IEEE Personal Communications* 8(4) 2001, 66-71.

25. Shu, N.C. Visual Programming: Perspectives and Approaches. *IBM Systems Journal 28*, 525-547, 1989.

26. Weiser, M. Computer for the 21st century. *Scientific American,* 265(3) 1991, 94-104.